LA TUA RIVISTA RASPBERRY PI **UFFICIALE**

The MagPi

La rivista ufficiale Raspberry Pi in italiano, da RaspberryItaly.com

Numero 54 Febbraio 2017 www.raspberr...

GUIDA ALLA LINEA DI COMANDO

Sudo apt-get good!

PROGRAMMAZIONE ORIENTATA AGLI OGGETTI

Scopri la programmazione orientata agli oggetti con Python

REALIZZA UNO SPECCHIO MAGICO

lo stanno plasmando perchè sia uno dei gadget più caldi del 2017
Realizzalo oggi, con la guida passo - passo.

Tuesday, February 26, 2017
15:10 18

Gratuito!

The MagPi magazine is published by Raspberry Pi (Trading) Ltd., Mount Pleasant House, Cambridge, CB3 0RN. ISSN: 2051-9982

L'UNICA RIVISTA PI SCRITTA DALLA COMUNITÀ RASPBERRY PI

REALIZZA IL TUO SPECCHIO MAGICO

Vivi il futuro con lo specchio connesso in rete, uscito direttamente dalla fantascienza

Viviamo in un mondo in cui molti oggetti stanno diventando parte dell'IOT (Internet delle Cose). Frigoriferi che inviano mail perché è finito il latte. Lavatrici che ti avvisano via Twitter quando il lavaggio è finito. Un'applicazione sullo smartphone per accendere le luci. Molte di queste cose possono apparire effimere, però; e probabilmente è questo il motivo per cui il concetto del Magic Mirror, ha avuto tanto seguito nella comunità dei maker: mostrarti le informazioni utili mentre ti specchi prima di uscire di casa. Uno strumento passivo ed utile. Michael Teeuw ha portato per primo il concetto sul Raspberry Pi rendendone facile la realizzazione a chiunque. Tutto quello di cui hai bisogno per costruirne uno è un po' di tempo libero ed una sega affilata.

IL PROCESSO
I PASSI PER COSTRUIRE UNO SPECCHIO MAGICO

OTTIENI I PEZZI
Tutto quel che ti serve per realizzare il tuo specchio personalizzato

16

MONTA LO SPECCHIO
Come mettere assieme la cornice e l'elettronica

18

TROVI TUTTO IL SOFTWARE E LE INFO DI CUI HAI BISOGNO SU:

MAGICMIRROR.BUILDERS

LA PARTE FURBA DEL CODICE

Il codice per lo specchio è tutto qui, pronto per essere installato con una sola linea

SPECCHIO A DUE VIE

Se lo fai nel modo giusto, puoi realizzare uno specchio ricco di informazioni

COSTRUIRE LA CORNICE

Tira fuori il metro e la matita, e datti da fare

PROGRAMMA LO SPECCHIO

Il codice è già qui: ecco come inserirlo nel tuo specchio.

22

PERSONALIZZA LO SPECCHIO

Fai alcune modifiche e personalizza il tuo nuovo specchio

24

COMPONENTI

Quel che ti serve per realizzare il tuo specchio

MATERIALI DI COSTRUZIONE

LEGNO PER IL FRONTALE

La parte anteriore si può realizzare con legno per battiscopa o modanatura – assicurati che sia più largo del legno che utilizzerai per la cornice in modo che riesca a contenere tutto.

CHIODI

Li useremo per fissare il frontale; 15-20mm dovrebbero essere sufficienti.

LEGNO PER LA CORNICE

Il compensato è una buona scelta. Se non sei proprio alle prime armi con la falegnameria potresti tuttavia usare del pino o altri materiali. Assicurati che sia robusto ed abbastanza spesso per contenere tutta l'elettronica che serve – dai un occhiata ai passi per costruirlo per farti un idea delle misure necessarie.

VITI

Servono alcune viti da legno per costruire la cornice. Non devono essere enormi, basta che siano 20mm più lunghe dello spessore del legno.

VERNICE E SIGILLANTE

Di sicuro vorrai che la cornice abbia un bell'aspetto, quindi usa del turapori o stucco per renderlo più liscio ed della vernice per completarlo. Se ti piace il legno che hai scelto puoi utilizzare del mordente.

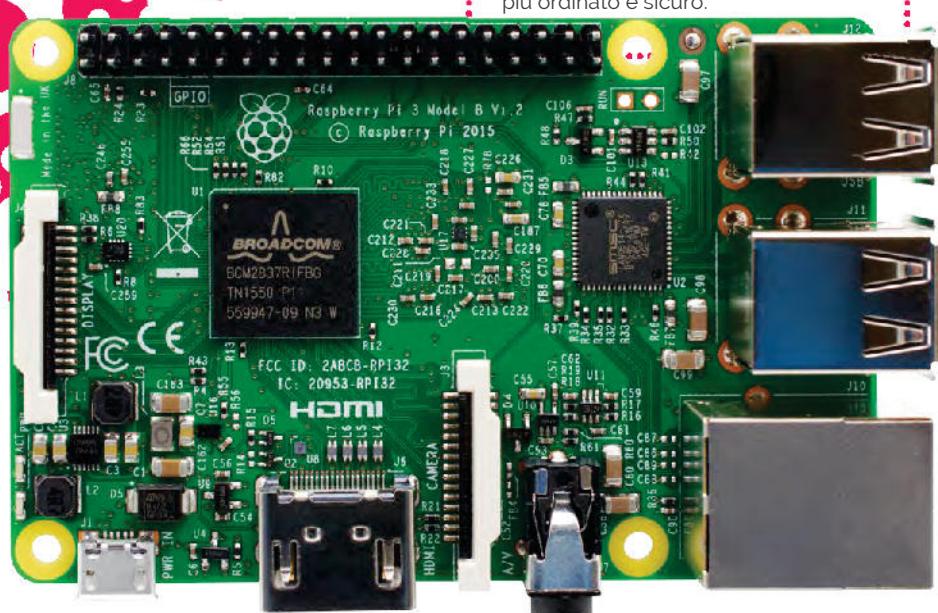
WOOD GLUE

COLLA PER IL LEGNO

La useremo per tenere insieme tutti i pezzi. Considerala un'aggiunta di sicurezza alle viti.

FALSO SPECCHIO

Comprane uno della stessa misura del monitor, che sarà anche la misura della cornice. Ne puoi trovare anche fatti di acrilico.

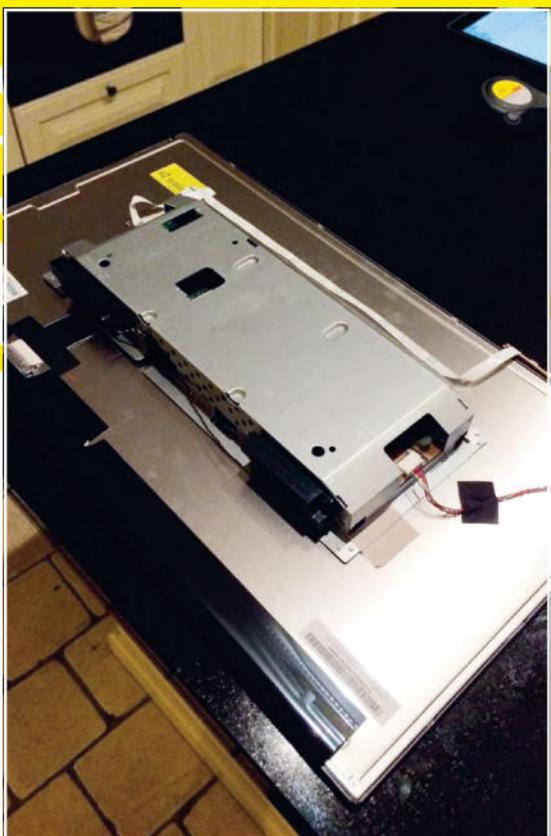
ELETTRONICA

MONITOR/TELEVISORE

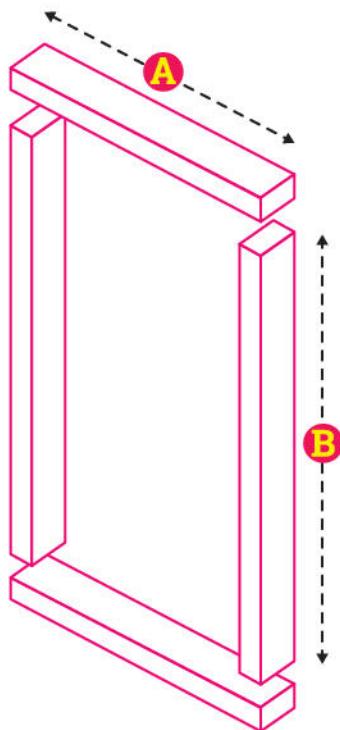
Un generoso monitor o un vecchio televisore LCD dovrebbero essere perfetti per questo. Più leggero è, e meglio è, però. Puoi decidere di rimuovere il contenitore esterno del monitor, se è possibile, per risparmiare spazio e peso.

RASPBERRY PI

Naturalmente ne servirà uno da collegare al televisore. Un case (e un dognle WiFi se userai un Raspberry Pi2 o precedenti) dovrebbe contribuire a rendere il tutto un po' più ordinato e sicuro.

CAVI

Ti serviranno i cavi di alimentazione per il monitor e il Raspberry Pi, e naturalmente il cavo HDMI per l'uscita video.

ATTREZZI

Ti servirà una sega, un martello, un trapano, alcuni morsetti e diversi strumenti da pittura, per questo progetto. Recuperali e tienili pronti.

PASSO 01:

MISURARE IL MONITOR

Prendi le misure dello schermo. Anche lo spessore è importante per assicurarti che il legno scelto vada bene; è preferibile avere un piccolo spazio tra il monitor ed il muro, quindi assicurati che lo spessore del legno lo permetta.

PASSO 02:

TAGLIARE IL LEGNO

La cornice di base è costituita da quattro pezzi di legno, i due lati e le fasce sopra e sotto. I lati saranno un po più lunghi degli altri in modo che lo specchio sia in posizione verticale. I lati devono avere la stessa lunghezza del lato lungo dello schermo, mentre le fasce sopra e sotto devono essere lunghe come il lato corto più lo spessore del legno in modo che possano essere uniti comodamente a formare un rettangolo. Cerca di non realizzare lo spazio per il monitor troppo stretto – lascia un millimetro o due extra.

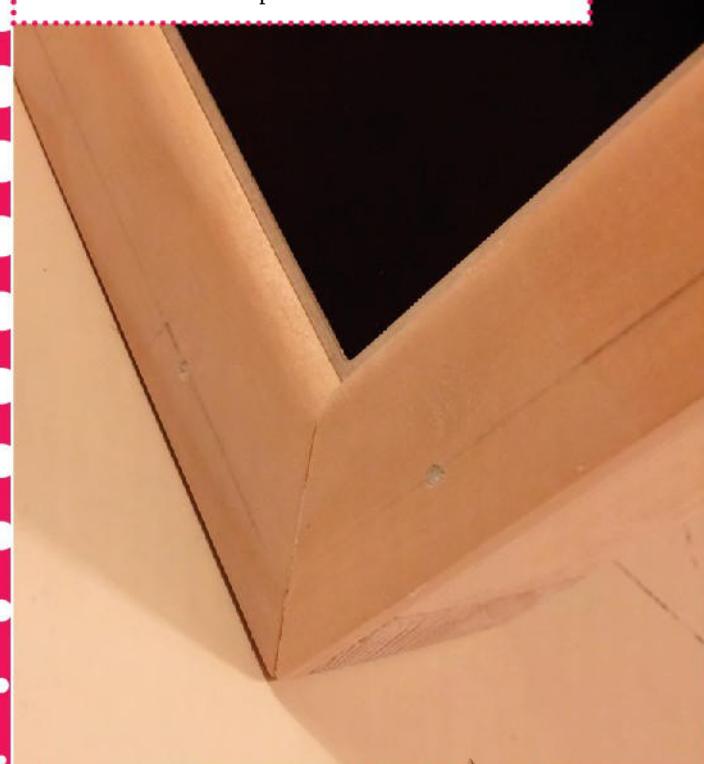
A Larghezza del monitor + spessore del legno × 2

B Altezza del monitor

RICORDA: MISURA DUE VOLTE, TAGLIA UNA VOLTA SOLA

PASSO 03:

ASSEMBLARE LA CORNICE

Questa è la cornice principale, quindi è meglio realizzarla il più robusta possibile. Avvitare il legno con due viti la renderà bella e solida, ma aggiungi un po' di colla da legno sarà per rendere la giunzione ancora più salda.

PASSO 04:

TAGLIA IL FRONTALE

Una volta realizzato il telaio principale, possiamo aggiunge la parte frontale della cornice. Ha due funzioni: è smussata sul monitor rendendolo un po' più bello esteticamente, e fa da profilo per per contenere il monitor e lo specchio in posizione. Assicurati che il legno sia più ampio di quello usato per il telaio, proprio per creare questo profilo, e taglia le estremità ad angolo per quando li unirai tra loro.

PASSO 05:

FISSA IL FRONTALE

Con attenzione, inchioda i pezzi frontali, facendo attenzione che siano a filo con il bordo esterno. Usa prima un chiodo a ogni estremità, per essere certo che sia orientato correttamente (non inchiodarlo tutto fino a che non sei sicuro), o usa dei morsetti per serrarli tutti assieme. Una volta posizionati un paio di pezzi, con gli altri due sarà molto più facile.

PASSO 06:

PARTI FINALI

Col trapano, realizza qualche foro sulla parte superiore e inferiore del telaio, come mostrato – servono per favorire la ventilazione. Non ci verrà tostato niente all'nterno, ma è meglio avere dell'aria che ci passa attraverso. Ti sarà utile montare anche una traversa, come mostrato, con delle scanalature per appendere il tutto a delle viti a tassello nella parete. Dovresti fare una piccola scanalatura sul retro del listello inferiore per far giungere i cavi di alimentazione fino al monitor e al Pi. Infine, crea delle piccole staffe, che puoi fissare al telaio per mantenere in posizione il monitor, evitando che cada dalla parte posteriore.

PASSO 07:

STUCCARE E LEVIGARE

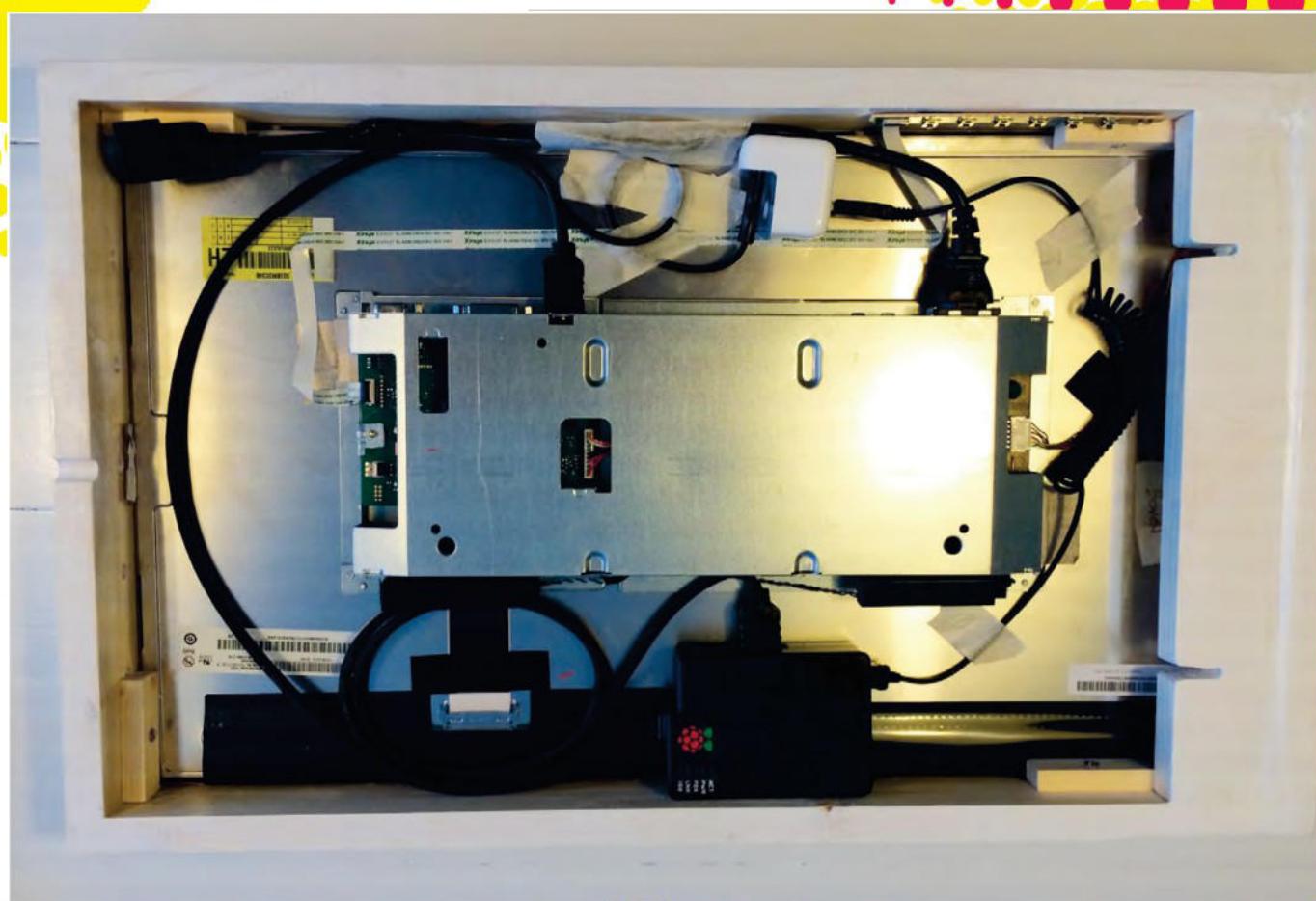
Utilizza un po' di stucco per appianare ogni avvallamento, comprese le teste dei chiodi sulla parte frontale. Poi leviga, per appianare tutto e preparare il legno per la verniciatura.

PASSO 08:

VERNICIATURA

In questo esempio, il telaio viene verniciato di bianco con una vernice per legno. Fallo in un locale ben areato.

PASSO 09:

INSERIRE LO SPECCHIO

Quando la vernice è asciutta, ribalta la cornice in modo che appoggi sulla parte frontale e con attenzione posaci dentro lo specchio. Lo terremo in posizione utilizzando la sporgenza della modanatura frontale e il monitor, creando a tutti gli effetti un sandwich in cui lo specchio fa la parte del prosciutto.

PASSO 10:

INSTALLARE L'ELETTRONICA

Questa parte è abbastanza semplice: basta mettere il monitor sul retro, fissarlo con le piccole staffe in legno, e collegare il Raspberry Pi a lla HDMI. Posa i cavi di alimentazione nella apposita scanalatura che hai creato.

PASSO 11: SUGGERIMENTO

UN CAVO, DUE DISPOSITIVI

Una cosa che il creatore maker più esperto potrebbe fare è abbinare il cavo di alimentazione del televisore con un alimentatore USB. In questo modo, avrai bisogno di far passare attraverso lo specchio un solo cavo.

PROGRAMMA IL TUO SPECCHIO

Installa il software sul tuo Raspberry Pi e rendi il tuo specchio assolutamente magico

Una volta appeso lo specchio al muro, o in qualsiasi altro posto ad esso destinato, è arrivato il momento di installare il software. Michael ha reso il processo molto semplice, e tutto quello di cui hai bisogno è digitare questi comandi:

```
curl -sL http://magpi.cc/MirrorInstall | bash
```

Procederà col processo di installazione e la definizione dei defaults ... ed ecco fatto! Lo specchio è pronto.

PERSONALIZZAZIONE

Bene, siamo quasi pronti: dai uno sguardo alle impostazioni predefinite per essere sicuro che tutto funzioni come desideri. Le impostazioni sono registrate in un file di configurazione che puoi creare così:

```
cp ~/MagicMirror/config/config.js.sample
~/MagicMirror/config/config.js
```

Adesso puoi aprilo e modificarlo con:

```
nano ~/MagicMirror/config/config.js
```

Queste sono alcune opzioni che puoi modificare...

Il file di configurazione di esempio ti consente di iniziare, ma c'è molto altro che puoi fare

OPZIONE	DESCRIZIONE
`port`	La porta utilizzata dal server MagicMirror. Quella predefinita è la 8080.
`address`	L'indirizzo IP dello specchio, lo useri per collegarti.
`ipWhitelist`	La lista degli IP autorizzati ad accedere allo specchio. Il valore predefinito: `["127.0.0.1", "::ffff:127.0.0.1", "::1"]`. Si può definire indirizzo e maschera di sottorete: (`["127.0.0.1", "127.0.0.1/24"]` oppure un intervallo di indirizzi IP (`["127.0.0.1", ["192.168.0.1","192.168.0.100"]]`).
`zoom`	Il fattore di zoom che determina la dimensione dell'interfaccia utilizza. Il valore di default è 1.0
`language`	La lingua dell'interfaccia. I valori possibili sono en, nl, ru, fr, e così via, ma il predefinito è en.
`timeFormat`	Il formato orario. I valori possibili sono 12 o 24. Il predefinito è 24.
`units`	Le unità di misura utilizzate per il meteo. I valori possibili sono metrico decimale o imperiale. Il predefinito è metrico decimale.
`modules`	Un elenco di moduli attivi. Ce ne deve sempre essere almeno uno attivo.
`electronOptions`	Una serie di opzioni per Electron (il browser). Questo permette di impostare ad esempio la dimensione e la posizione della finestra del browser (predefinito `.width = 800` e `.height = 600`). La modalità chiosco si può impostare inserendo `.kiosk = true`, `.autoHideMenuBar = false`, `.fullscreen = false`.

DATA
Che ore sono e che giorno siamo? Meglio smettere di ammirare il tuo riflesso o farai tardi al lavoro!

TEMPERATURA
Com'è il tempo? Farà bello oggi? O c'è tempo da ombrello? Meglio che rientri prima del tramonto

CALENDARIO
Oggi è festa? Si sta avvicinando una riunione? Questo colore si cravatta è adatto per la riunione?

CARINERIE
Grazie specchio, non sei niente male nemmeno tu! I miei complimenti alla fantaztica persona che ti ha realizzato

NEWS
Qualche notizia importante che richiede la tua attenzione? Come è il traffico? Oh, guarda: rifanno in tv il mio show preferito!

Tuesday, December 13, 2016
15:10
Christmas
New Year's Day
M L King Day
Valentine's Day
Presidents' Day
Good Friday
22:29
-0.6°
You look sexy!
Rex Tillerson, Exxon C.E.O., Chosen as Secretary of State

PERSONALIZZA IL TUO SPECCHIO

Rendi il tuo specchio veramente tuo, aggiungendo e personalizzando moduli

Oltre a quelli predefiniti puoi installare moduli di terze parti creati dalla comunità o da te. Sono facili da aggiungere: devi solamente scaricare i file e modificare la configurazione perché vengano utilizzati.

Prima di tutto dai un occhiata ai moduli nella pagina del repository GitHub di MagicMirror qui: **magpi.cc/2iqWPUh** .. Troverai una lista di ottimi moduli da installare, come il modulo per monitorare i Bitcoin o la vignetta del giorno da XKCD. Scegline uno che ti interessa e copia il link.

Per installare il modulo, prima spostati nella cartella **modules** con **cd ~/MagicMirror/modules** scarica i file necessari col comando:

```
git clone https://github.com/[author]/
[module-name]
```

...usando il link di GitHub che hai copiato prima. Controlla il file readme del modulo per verificare se ci sono delle altre operazioni da fare, in caso contrario, prima apri il file **config.js**, e poi aggiungi il nuovo modulo nella sezione dei moduli. Devi formattare il testo come in questo esempio:

```
{
  module: 'module name',
  position: 'position',
  header: 'optional header',
  config: {
    extra option: 'value'
  }
},
```

Qui accanto c'è l'elenco completo dei parametri utilizzabili...

NON VUOI SMANTELLARE IL TUO SPECCHIO?
Usa SSH da un altro computer per accedere al Pi

OPZIONE	DESCRIZIONE
`module`	Il nome del modulo. Può contenere anche la sottocartella. Ecco alcuni validi esempi **`clock`**, **`default/calendar`**, e **`modules/[module name]`**.
`position`	La posizione del modulo nello specchio. Alcuni possibili valori sono: **`top_bar`**, **`top_left`**, **`top_center`**, **`top_right`**, upper**`_third`**, **`middle_center`**, **`lower_third`**, **`bottom_lef`**, **`bottom_center`**, **`bottom_right`**, **`bottom_bar`**, **`fullscreen_above`**, e **`fullscreen_below`**. Questo campo è opzionale, ma molti moduli richiedono che venga impostato. Controlla la documentazione del modulo per maggiori informazioni. Più moduli configurati con la stessa posizione vengono visualizzati secondo l'ordine del file di configurazione. E' necessario fare un po' di prove.
`classes`	Classi aggiuntive che vengono passate al modulo. Campo facoltativo.
`header`	Per mostrare del testo introduttivo prima del modulo aggiungi questo campo opzionale.
`disabled`	Disabilita la creazione del modulo. Campo facoltativo.
`config`	Un oggetto con la configurazione del modulo. Controlla la documentazione del modulo per maggiori informazioni. Questo campo è facoltativo a meno che il modulo non richieda delle configurazioni aggiuntive.

SUPPORTO & COMMUNITY

Hai realizzato il tuo specchio, ora qui trovi alcuni modi per seguire il suo sviluppo

SITO MAGICMIRROR

magicmirror.builders

La prima tappa per le informazioni sullo specchio magico (MagicMirror) è la home page del software. Ci troverai diversi link a portata di mano, come quelli per il blog per gli aggiornamenti sul progetto, GitHub per il codice sorgente e le opzioni di configurazione avanzata dello specchio, così come il link al forum e ai moduli disponibili. È anche un buon modo per introdurre rapidamente un amico a questo progetto.

FORUM SUL MAGICMIRROR

magpi.cc/2je2dXI

La Community del MagicMirror vive qui, e le risposte a molti dei problemi che a lungo termine ti potresti trovare a dover affrontare, qui probabilmente hanno già trovato risposta. Sono un gruppo amichevole, tosto, quindi se non ti è possibile trovare una soluzione, puoi sempre fare una chiacchierata con loro per trovare che cosa potrebbe essere andato storto. Potrai anche essere in grado di confrontare le note su eventuali moduli che deciderai di creare, o qualsiasi altro piccolo aggiornamento o suggerimenti di costruzione potresti desiderare di conoscere.

MAGICMIRROR MODULE DEVELOPMENT DOCUMENTATION

magpi.cc/2jebu

Sogni di creare il tuo modulo personalizzato per il tuo specchio? Avrai bisogno di conoscere come funzionano le API e il codice, e lo puoi fare con la documentazione fornita sul repository GitHub di MagicMirror. Abbiamo visto alcuni ottimi add-on per il codice, comprese le variazioni stagionali e gli orari dei treni in tempo reale. Tutto quel che ti serve è una fonte da cui ottenere i dati!

MagicMirror² Module Development Documentation

This document describes the way to develop your own MagicMirror² modules.

Module structure

All modules are loaded in the `modules` folder. The default modules are grouped together in the `modules/default` folder. Your module should be placed in a subfolder of `modules`. Note that any file or folder your create in the `modules` folder will be ignored by git, allowing you to upgrade the MagicMirror² without the loss of your files.

A module can be placed in one single folder. Or multiple modules can be grouped in a subfolder. Note that name of the module must be unique. Even when a module with a similar name is placed in a different folder, they can't be loaded at the same time.

Files

- **modulename/modulename.js** - This is your core module script.
- **modulename/node_helper.js** - This is an optional helper that will be loaded by the node script. The node helper and module script can communicate with each other using an intergrated socket system.
- **modulename/public** - Any files in this folder can be accesed via the browser on `/modulename/filename.ext`.
- **modulename/anyfileorfolder** Any other file or folder in the module folder can be used by the core module script. For example: *modulename/css/modulename.css* would be a good path for your additional module styles.

PASSA ALLA COMMAND LINE

Utilizzando la Linea di Comando, puoi lavorare meglio e più velocemente. Scopri oggi come puoi cominciare...

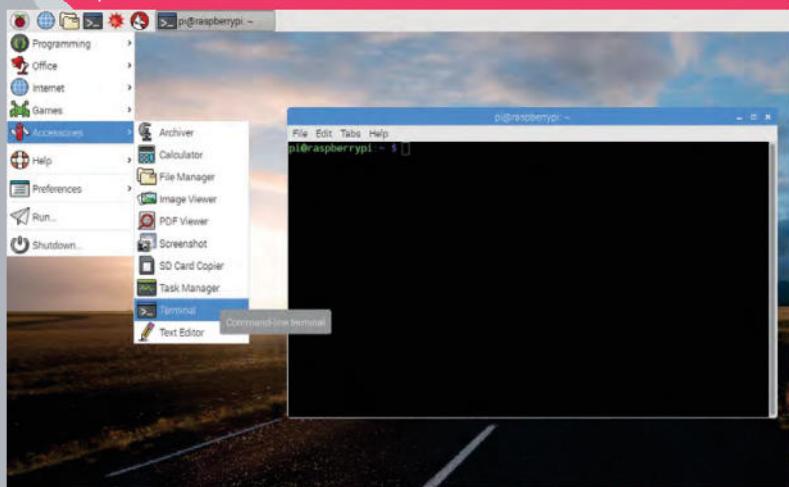
Cosa Serve

- Raspberry Pi
- Raspbian con PIXEL

A meno che tu non sia cresciuto nel 1980 o anche prima, con tutta probabilità sei abituato a usare solo le GUI (graphical user interfaces, interfacce grafiche utente) e gli ambienti desktop.

In realtà non c'è nulla di male nelle GUI, e Raspbian ne possiede una particolarmente piacevole chiamata PIXEL.

Ma al di sotto alle icone esiste tutto un altro mondo: la linea di comando. È lì che c'è il tuo vero computer. Con la linea di comando, non sei limitato a fare quello che le applicazioni desktop ti consentono di fare. Puoi fare qualsiasi cosa sul tuo computer, e farlo molto più velocemente.

Pensa sia come guidare una macchina. Se hai sempre e solo usato la GUI, allora stai guidando col cambio automatico. La linea di comando è come passare al cambio manuale. È molto più complicato, ma ottieni molto più controllo e senti di guidare in modo più autentico.

La linea di comando uò essere scoraggiante per i nuovi arrivati, ma in realtà non è così. Con pochi comandi, la potrai padroneggiare.

Scrivere i comandi

Quando avvii un Raspberry Pi, automaticamente ti ritrovi all'interno dell'interfaccia desktop PIXEL.

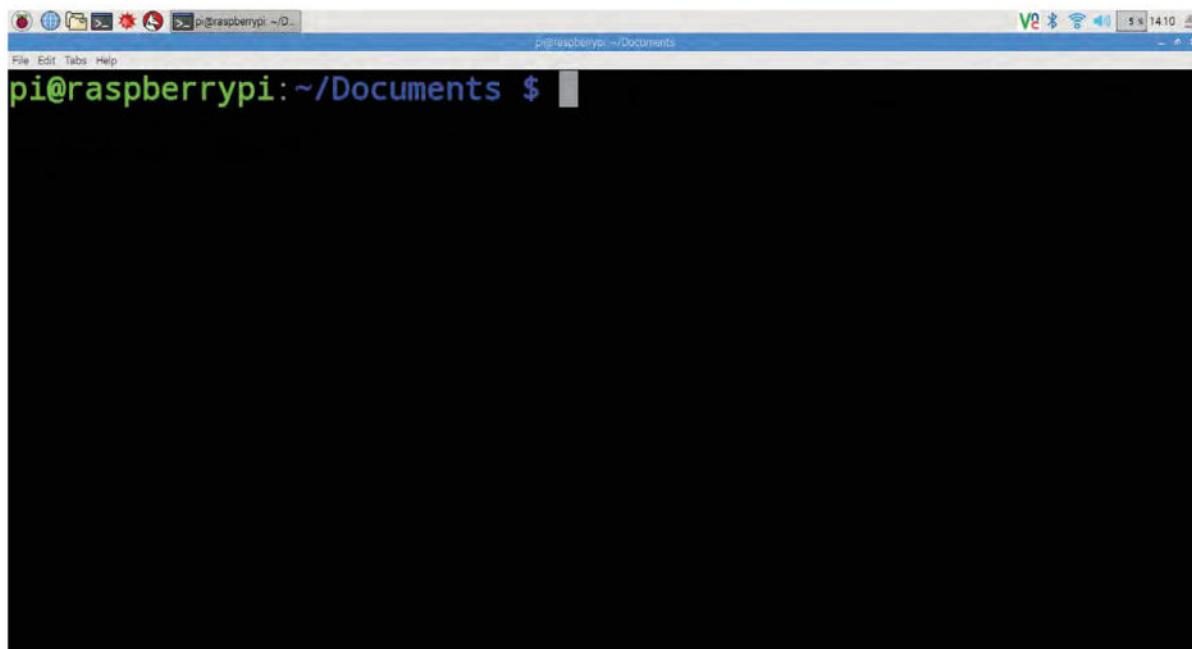
Il modo più veloce per avere'accesso alla linea di comando è attraverso il Terminale.

Molte persone accedono alla linea di comando attraverso il Terminale sul desktop PIXEL.

Clicca sull'icona del Terminale in alto nella barra dei menu (o seleziona Menu > Accessories > Terminal). Si apre una finestra con lo sfondo nero e del testo verde e blu. Vedrai il prompt dei comandi:

```
pi@raspberrypi:~ $
```

Ora sei nella linea di comando. Immetti i comandi utilizzando una interfaccia testuale. Digita **echo Ciao Mondo** e premi ***RETURN***, vedrai 'Ciao Mondo' apparire. Sotto, vedrai ancora il prompt **$**, pronto ad accettare un altro comando.

Molti utenti accedono alla linea di comando tramite il Terminale, ma c'è un altro modo, conosciuto come "console virtuale". Premi ***CTRL+ALT+F1*** e il desktop sparirà. Apparirà invece uno schermo nero, che mostrerà la scritta 'Raspbian o (Debian) GNU/Linux 8 raspberry pi tty' e sotto di essa, 'raspberrypi login'.Se tu non sei automaticamente loggato, immetti ***pi*** e premi ***RETURN***, poi inserisci la tua password (***raspberry*** è quella di default).

Ora puoi usare la linea di comando tutto schermo. Puoi ritornare alla schermata di PIXEL, utilizzando ***CTRL+ALT+F7*** e poi tornare alla console virtuale usando ***CTRL+ALT+F1***. Console virtuali aggiuntive potranno essere aperte usando ***CTRL+ALT+F2***, fino a ***F6***. Ognuna di esse ha il proprio login e opera in modo indipendente.

Se preferisci la linea di comando, puoi avviare direttamente Raspbian con essa, invece che con il desktop PIXEL. Apri Raspberry Pi configuration (Menu > Preferences > Raspberry Pi configuration). Modifica l'impostazione di avvio a "To CLI" e clicca OK . Ora, una volta riavviato, partirà dalla linea di comando (scrivi **startx** per avviare il desktop di PIXEL)

Individua dove sei

La prima cosa che hai bisogno di imparare è come scoprire dove sei. DI default, sei nella tua cartella home. Scrivi il seguente comando e premi ***RETURN***:

```
pwd
```

pi@
La prima parte della linea di comando è il tuo nome utente, seguito dal simbolo @. Puoi verificarlo sulla linea di comando, immettendo `whoami`.

raspberrypi
Dopo la @, arriva il tuo host name. Che è il nome del tuo computer: 'raspberrypi' è il nome di default.

~/Documents
Dopo l'host name è presente la directory di lavoro corrente. Viene mostrato semplicemente '~' quando sei nella tua directory home.

$
Il simbolo del Dollaro, indica che stai operando come normale utente.

Questo comando è l'abbreviazione di "print working directory" (stampa directory di lavoro) e ti dice dove sei. La linea di comando restituirà **`/home/pi`**.

La cartella home è la stessa che appare di default quando apri l'applicazione File Manager. È possibile visualizzare i file e le directory all'interno della cartella di lavoro, usando il comando list (**`ls`**):

```
ls
```

Così vedrai le stesse directory (o cartelle) che vedi in File Manager: ***Desktop***, ***Downloads***, ***Documents***, e così via.

Il percorso del file

Prima di proseguire nelle directory, è necessario comprendere il percorso del file e la differenza tra un percorso 'relativo' e 'assoluto'.

I file vengono collocati nelle cartelle (che sono chiamate 'directory' nella linea di comando). Nelle GUI visuali, puoi vederle come cartelle, e aprendole rivelano altri file e cartelle. La tua home directory contiene una directory ***Documents***, e al suo interno ci sono altre tre directory: ***Blue J Projects***, ***Greenfoot Projects***, ***e Scratch Projects***.

/home/pi/Documents/Scratch\ Projects

Nel percorso del file qui sopra, la prima barra è la radice del tuo disco fisso. Qui hai la directory chiamata ***home***, che contiene tutti gli utenti. Dentro di essa c'è un'altra directory chiamata ***pi*** (che sei tu), e dentro questa c'è un'altra directory chiamata ***Documents***, che a sua volta contiene quella chiamata ***Scratch Projects***.

Il vostro occhio di falco avrà senz'altro notato una strana barra al contrario (backslash: '\') . Non puoi avere avere spazi nei nomi dei file, così, utilizzi un backslash seguito da uno spazio, all'interno della

CAPIRE IL LINGUAGGIO

C'è molta confusione relativa al gergo che ruota attorno alla linea di comando. Termini come linea di comando, shell, e terminale sono spesso usati come sinonimi.
Ognuno ha un significato preciso...

- **TERMINALE:** Questo è il programma che utilizzi per accedere alla linea di comando dal desktop PIXEL in Raspbian (il suo nome completo è LXTerminal).
- **CONSOLE:** Questo è lo schermo terminale con tastiera fisico. E' una specie di computer vuoto, utilizzato per connettersi a grandi computer mainframe.
- **CONSOLE VIRTUALE:** Queste sono le versioni virtuali di una console fisica. In Linux, si dispone di console virtuali multiple accessibili tramite i tasti CTRL+ALT e i tasti funzione.
- **TTY:** Teletypewriter. In Linux, tty è usato per mostrare quale console virtuale stai utilizzando: tty1, tty2, e così via.
- **LINEA DI COMANDO:** Questo è l'ambiente testuale in generale o la riga specifica su cui stai lavorando. La linea di comando inizia con il segno del dollaro ($), conosciuto come 'prompt'.
- **SHELL:** È un'interprete della linea di comando. Circonda il Kernel del computer (da qui il nome shell: conchiglia). Per arrivare al Kernel, si passa attraverso la Shell. Essa interpreta i tuoi comandi testuali e li trasforma in codice che il Kernel possa capire.
- **BASH:** Sta per 'Bourne Again Shell' ed è un tipo di shell utilizzato da Debian (la versione di Linux su cui è basato Raspbian).

ls
La prima parte del comando è il comando stesso. Qui abbiamo ls, che elenca il contenuto di una directory

-lah
Dopo il comando, vengono le opzioni. cominciano con un trattino e sono tipicamente singole lettere. Ognuna modifica il comando. Qui abbiamo 'l', 'a' e 'h'. Stanno per la modalità lunga, tutti i file, e in formato più leggibile.

/home/pi/Documents
La parte finale del comando è l'argomento. Spesso è un nome di file o un percorso. Qui stiamo listando un percorso assoluto (percorso diretto alla directory Documents). Se ometti l'argomento, visualizzerà i contenuti della directory corrente.

```
pi@raspberrypi:~ $ ls -lah /home/pi/Documents/
total 20K
drwxr-xr-x  5 pi pi 4.0K Dec 15 14:37 .
drwxr-xr-x 31 pi pi 4.0K Dec 15 12:52 ..
-rw-r--r--  1 pi pi    0 Dec 15 14:37 a_file.txt
drwxr-xr-x 11 pi pi 4.0K Nov 25 17:49 BlueJ Projects
drwxr-xr-x  5 pi pi 4.0K Nov 25 17:49 Greenfoot Projects
drwxr-xr-x  2 pi pi 4.0K Nov 25 17:46 Scratch Projects
pi@raspberrypi:~ $
```

linea di comando. La maggior parte delle volte puoi usare tasto ***TAB*** per completare rapidamente nomi di file lunghi (vedi completamento TAB').

Come già detto, i percorsi dei file sono di due tipi: relativo e assoluto. I percorsi relativi sono 'relativi' alla tua directory di lavoro, che è ***/home/pi/*** all'inizio. Se scrivi **ls** da solo, viene mostrato il contenuto della directory corrente. Puoi visualizzare il contenuto di una directory dentro la tua directory corrente usando **ls** e il suo nome:

```
ls Documents
```

Puoi vedere il contenuto della directory al livello superiore utilizzando due punti (..):

```
ls ..
```

Questo comando visualizza i file relativi al punto in cui ti trovi nel file system. Se ti sei spostato nella cartella ***Downloads*** e poi hai digitato **ls Documents**, otterrai un errore, perché non ci sono directory ***Documents*** all'interno della cartella ***Downloads***.

Un percorso assoluto, d'altra parte, inizia sempre con una slash '/', che è la directory principale, la radice (la base del tuo disco fisso).

Digita:

```
ls /
```

...per visualizzare la directory principale (root). Qui vedrai tutte le directory e i file che compongono Linux. Vedrai directory come ***bin*** (per i binari), ***boot*** (file usati per avviare il sistema), e ***home***, che contiene la tua cartella utente.

Digita:

```
ls /home/pi
```

...e vedrai il contenuto della tua cartella home, proprio come se tu avessi digitato **ls** all'interno di essa.

Puoi utilizzare i percorsi assoluti senza che importi quale sia la tua directory corrente, perché essi partono sempre dalla radice.

Muoversi attorno

Fino ad ora siamo rimasti nella cartella home e abbiamo dato un'occhiata intorno usando **ls**. Ti sei mosso da una directory a un'altra usando il comando **cd** (change directory):

```
cd Documents
```

Ora digita:

```
pwd
```

...vedrai un differente percorso di lavoro: /home/pi/Documents. Per tornare indietro di una directory (conosciuta come directory 'padre'), usa due punti.

```
cd ..
```

La linea di comando può essere usata per gestire i file e le directory sul tuo sistema

Digita **pwd** ancora e tornerai nella cartella home. Ora prova ad usare un percorso assoluto. Digita:

```
cd /
```

...e sarai nella directory di root. Digita **ls** per vedere le cartelle alla base del tuo disco fisso. Immetti:

```
cd /home/pi
```

...per tornare alla cartella home. C'è una scorciatoia per questo:

```
cd ~
```

Il carattere tilde (~) è una scorciatoia per arrivare alla cartella home. Lo puoi usare anche all'inizio di un percorso assoluto. Ad esempio digitando:

```
cd ~/Downloads
```

...ti sposti nella tua cartella ***Downloads***, non ha importanza dove ti trovi nel sistema.

File

In tutto il filesystem troverai diversi tipi di file. Una bella selezione è nella cartella ***python_games***, quindi digita:

```
cd ~/python_games
ls -l
```

L'opzione **-l** seleziona la modalità 'lista lunga', la quale permette di visualizzare molte informazioni aggiuntive per ogni oggetto:

```
-rw-rw-r-- 1 pi pi 973 Jan 27 2015
4row_arrow.png
```

Da sinistra a destra, ogni oggetto è:

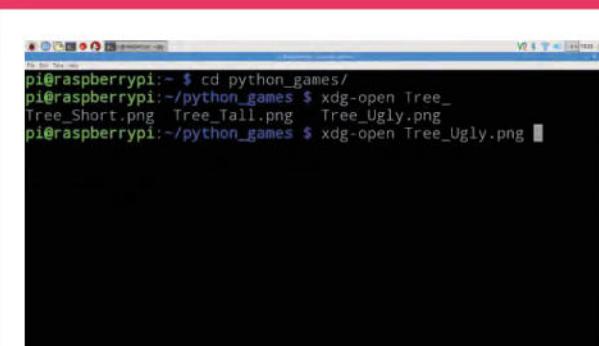
- ***Permessi:*** Gli utenti e i gruppi che possono accedere ad un file.
- ***Hard links:*** Il numero di file che sono linkati a questo file.
- ***Proprietario:*** La persona che possiede il file. Di solito ***pi*** o ***root***.
- ***Gruppo:*** Il gruppo al quale il file appartiene.
- ***File size:*** La dimensione del file.
- ***Modifica:*** L'ultima volta che il file è stato modificato.
- ***File name:*** Il nome del file.

La parte più difficile da interpretare è la lista di lettere e trattini che costituiscono i permessi. La prima lettera sarà a '-' oppure a 'd' a seconda se è un file o una directory. Il nostro ***4row_arrow.png*** è un file, quindi è un '-'.

Dopo di quello ci sono 9 lettere divise in gruppi di tre (vedi ***Fig 1 sopra***):

- ***Proprietario:*** Di solito questa è la persona che ha creato l'account.
- ***Gruppo:*** Questo è un gruppo di utenti. C'è solo un gruppo di default, **pi**, che contiene solo un utente (anche questo **pi**).
- ***Altri:*** Questi sono utenti di altri sistemi.

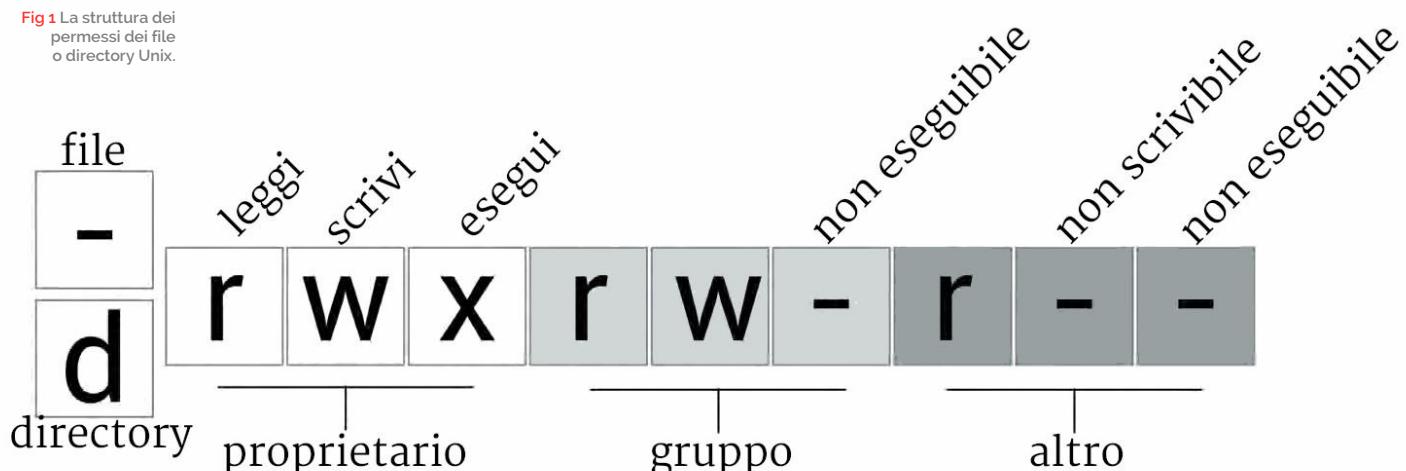
Ogni gruppo contiene le lettere 'rwx'. Queste lettere sono sempre in questo ordine e possono essere sia lettere che trattini. Una lettera indica che l'utente o il gruppo o altro ha accesso a leggere, scrivere o eseguire il file. Un trattino invece indica che non hanno quel livello di accesso.

Alcuni esempi includono:

- ***rwx*** legge, scrive, esegue
- ***rw-*** legge, scrive, ma non esegue
- ***r-x*** legge ed esegue
- ***r--*** solo lettura

Adesso che hai scoperto come muoverti all'interno della cartella di sistema dalla riga di comando, è tempo di imparare cosa altro puoi fare.

Prendi il comando

Uno dei primi comandi che hai bisogno di imparare è **mkdir**. Questo significa 'make directory' (crea directory). Spostati sulla cartella home e crea una directory chiamata test:

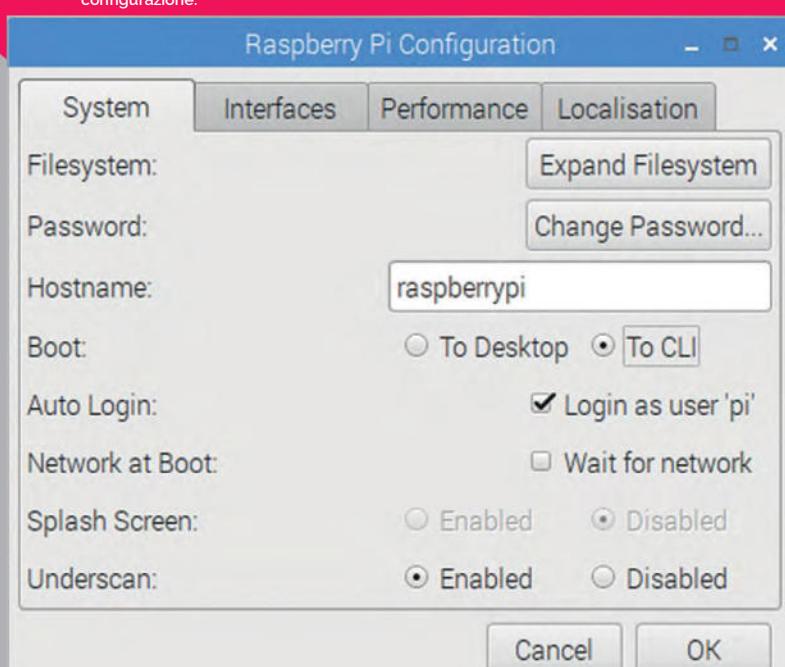
```
cd ~
mkdir test
cd test
```

COMPLETAMENTO **CON TAB**

Il trucco più utile da imparare per la riga di comando è il completamento con il tab. Premendo TAB in qualsiasi momento quando stai digitando un percorso, questo verrà completato con il nome del file o della directory per te. Usa **cd python_games** e digita **xdg-open Tr**, poi premi il pulsante TAB. Noterai che verrà completato con '**xdg-open Tree_**'. Ci sono tre file che iniziano con Tree, premi velocemente due volte TAB per vederli: **Tree_Short.png**, **Tree_Tall.png** e **Tree_Ugly.png**. Premi **S**, **T**, o **U** e poi premi ancora **TAB** per completare l'intero nome. Premi **RETURN** per aprirlo.

Il completamento con TAB è preziosissimo quando digiti nomi lunghi con molte lettere, numeri e segni di punteggiatura.

Fig 1 La struttura dei permessi dei file o directory Unix.

Per creare file userai un commando piuttosto strano chiamato **touch**. Ufficialmente, touch viene usato per aggiornare l'orario dell'ultima modifica di un file (lo raggiunge e lo 'tocca'). Se tocchi un file, l'orario corrente sostituisce quello visualizzato precedentemente.

Poche persone usano **touch** per quello, però. Un comodo effetto secondario di questo comando è che se tu tocchi un file che non esiste, in realtà crei un file vuoto. Digita:

```
touch test.txt
```

Creerai un file vuoto chiamato ***test.txt***. Digita **ls -l** E vedrai il nuovo file con tutti i suoi dettagli. Nota che la dimensione del file è 0. Questo perché il file è completamente vuoto.

Puoi modificare il contenuto del file usando un editore di testi chiamato nano:

```
nano test.txt
```

Puoi impostare Raspbian all'avvio dalla riga di comando invece che dalla interfaccia grafica nelle impostazioni di configurazione.

Puoi digitare e modificare il testo in nano ma i pulsanti Salva ed Esci differiscono dai tradizionali ***CTRL+S***, e ***CTRL+W***. Digita una linea singola, "Ciao Mondo!" e premi ***CTRL+O*** seguito da ***ENTER*** per salvare il file. Ora premi ***CTRL+X*** per uscire.

Digita **ls -l** ancora e noterai che la dimensione del file è cambiata da 0 a 12. Ogni lettera (spazio incluso) e 'a capo' alla fine (puoi vedere questo carattere con **od -c test.txt** se sei curioso), concorre alla dimensione.

Ora proviamo a cancellare il file. Questo comando rimuove
il file:

```
rm test.txt
```

Ora spostati sulla directory padre e usa un altro comando, **rmdir**, per rimuovere la directory vuota ***test***.

```
cd ..
rmdir test
```

Sfortunatamente, utilizzerai raramente rmdir per rimuovere directory perché di solito hanno file al loro interno. Puoi vedere il messaggio di errore con questi comandi:

```
mkdir test
touch test/test_file.txt
rmdir test
```

Ti dirà ***rmdir: failed to remove 'test': Directory not empty***. La soluzione è usare **rm** con l'opzione **-R**. Questa opzione sta per 'ricorsiva' cioè entra in ogni directory e sottodirectory e rimuove ogni singolo file e directory. Fai attenzione quando usi **rm -R**, perché le modifiche sono permanenti e viene cancellato tutto quello che si trova all'interno. Digita:

```
rm -R test
```

La directory ***test*** e tutto il suo contenuto scomparirà. Diversamente dall'ambiente desktop, nella riga di comando non esiste il Cestino. Quando rimuovi i file, questi saranno rimossi immediatamente e per sempre.

Opzioni

Molti comandi hanno delle opzioni che modificano il loro modo di operare. È usuale usare queste tre opzioni con il comando **ls:**

```
ls -lah
```

Le opzioni iniziano con un singolo trattino '-' seguito da una lettera per ogni opzione. Le tre opzioni usate sono:

- l = formato lista dettagliata
- a = tutti i file inclusi quelli nascosti
- h = leggibile da umani (rende i file di grandi dimensioni ,più leggibili)

Le opzioni tengono conto del maiuscolo o minuscolo. Quindi **ls -l e ls -L** sono due cose diverse ('l' minuscola è per il formato lista dettagliata mentre 'L' maiuscola significa senza referenze). A volte le opzioni sono scritte in forma estesa. Queste iniziano con due trattini e hanno un trattino solo come spaziatore. Questo comando è lo stesso di **ls -lah**:

```
ls -l --all --human-readable
```

Ma è più comune vedere (ed usare) le lettere singole.

Sudo

Sudo significa 'utente sostituto fa', anche se è spesso chiamato 'superuser fa'. Se hai più di un utente può Essere usato per eseguire i comandi come se tu fossi un altro utente.

È usato principalmente per ottenere accessi di root sul tuo sistema Linux. C'è un account che controlla il tuo utente Pi chiamato 'root'. Questo è un account con tutti i permessi e può apportare qualsiasi modifica al tuo sistema.

Il tuo account di default può vedere i file nella radice del tuo disco fisso, ma non può crearne di nuovi o cancellare quelli già esistenti. Digita:

```
cd /
touch test.txt
```

Vedrai *touch:* **cannot touch 'test.txt':** ***Permission denied***. Però, se digiti:

```
sudo touch test.txt
```

...viene creato il file ***test.txt*** nella root del tuo disco fisso. Puoi vederlo con **ls -l**.

Ora prova ad eliminarlo:

```
rm test.txt
```

Ti dirà ***rm: remove write-protected regular empty file 'test.text'?*** Premi **Y** e ti dirà ***rm: cannot remove 'test.txt': Permission denied***.
Dovrai usare sudo per rimuovere il file:

```
sudo rm test.txt
```

È facile comprendere che sudo è uno strumento davvero potente. Senza di lui non potresti installare software usando apt o apt-get. Ma usandolo, puoi rimuovere o cancellare importanti file di sistema. Digita **ls /bin** e vedrai molti programmi (conosciuti come binari) usati da Linux. Questi includono anche il comando **ls** che hai appena usato. Cancellare per errore questi file potrebbe rendere Il tuo sistema instabile.

Quindi utilizza sudo con attenzione. In Raspbian non è necessario digitare la password per usare sudo. In molti altri sistemi Linux ti verrà chiesta la password prima di poter utilizzare sudo.

Che succede, man?

Ci sono molti modi per chiedere aiuto tramite la linea di comando. Il primo a cui dovresti fare riferimento è **man**. Man sta per 'manuale' e ti fornisce le istruzioni per usare i comandi e gli strumenti di Linux. Digita:

```
man ls
```

...e vedrai il manuale per il comando list.
Nota che sotto la SINOSSI dice:

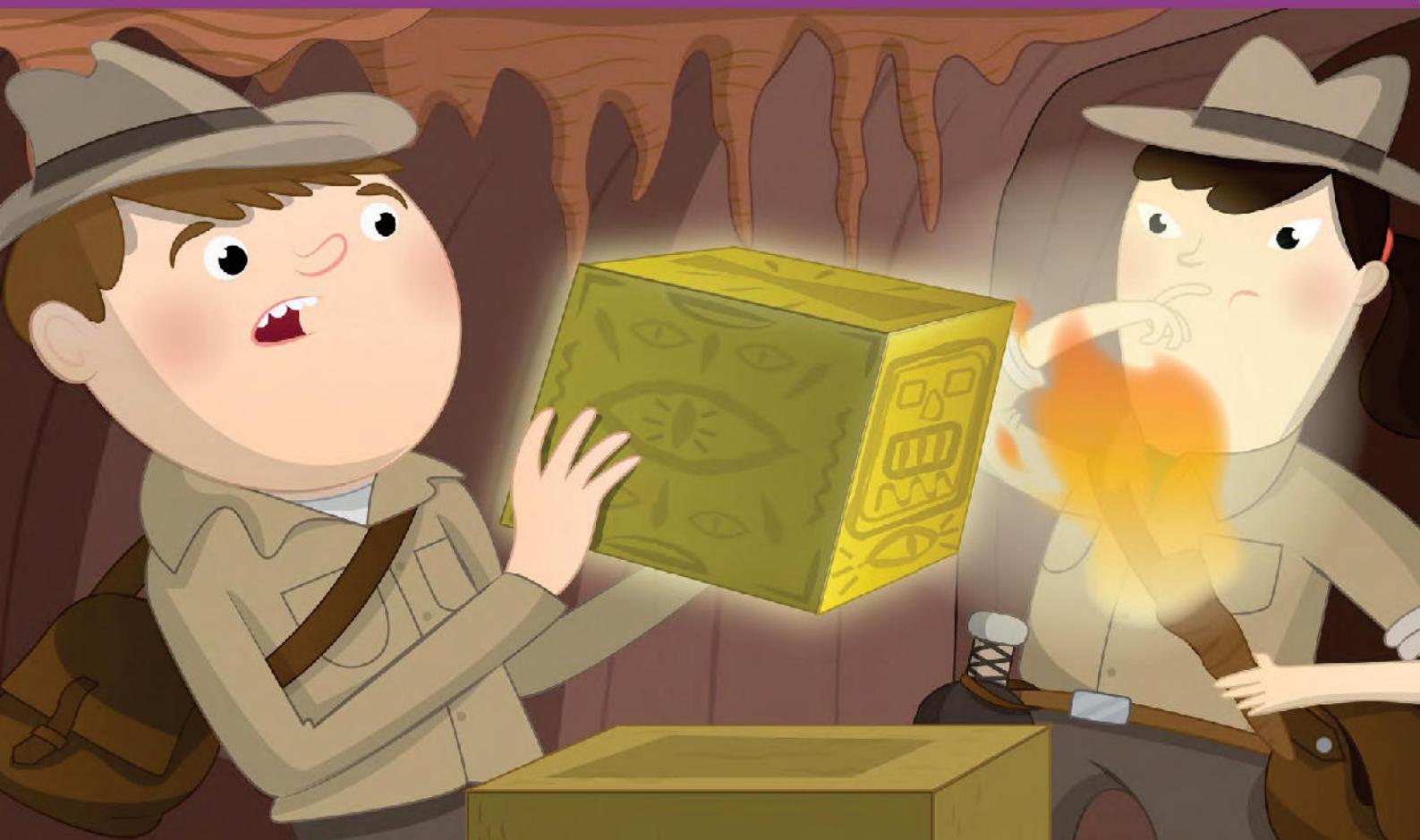
```
ls [OPTION]... [FILE]...
```

Questo ti mostra la struttura del comando. Quasi tutti i comandi posseggono la struttura 'comando, opzione, argomento' anche se alcuni argomenti hanno più di un argomento [FILE] (come una copia, che richiede un file di origine ed uno di destinazione).

Premi la barra spaziatrice per scorrere le istruzioni. Qui vedrai la lista di opzioni disponibili. Con **man**, puoi avere informazioni dettagliate su ogni strumento nella riga di comando. Puoi anche ottenere il manuale del comando **man** con:

```
man man
```

Se hai bisogno di un promemoria veloce su come utilizzare un comando, prova ad usarlo con **-h** o **--help** come opzione:

```
touch -help
```

...ti dirà quali opzioni sono disponibili con il comando **touch**. Puoi usarlo con molti comandi per rinfrescarti la memoria velocemente su come utilizzarli.

Spostarsi dalla GUI alla riga di comando è un'abilità vitale per hacker e programmatori. Qualsiasi cosa nel tuo computer, da un programma alle preferenze, è memorizzata nel file system, da qualche parte. Imparare ad utilizzare la riga di comando ti rende un utilizzatore di Raspberry Pi più capace.

Quindi la prossima volta che devi spostare un file o cancellare qualcosa non utilizzare il File Manager. Apri il terminale e compi l'operazione da riga di comando. Presto ti diventerà naturale.

COMPRENDERE LA PROGRAMMAZIONE ORIENTATA AGLI OGGETTI

Maneggia OOP usando Scratch e Python per creare gli stessi programmi

Nel mondo moderno, quasi tutto il codice che incontrerai è scritto con uno stile chiamato 'object oriented programming' (programmazione orientata agli oggetti) spesso abbreviato in OOP.

Se sei cresciuto con l'OOP questo risulta di sicuro il modo più naturale per programmare un computer.

Nel paradigma OOP il codice viene usato per creare oggetti; questi rappresentano oggetti reali del nostro mondo: un cane, una sedia o la ruota di un'auto.

Gli oggetti contengono sia le caratteristiche di una data cosa: l'altezza, l'età ed il nome di una persona ad esempio; sia le operazioni che l'oggetto può compiere: un cane può saltare, camminare o correre; una ruota può girare.

OOP combina le caratteristiche e le funzioni di un oggetto. In questo paradigma è molto facile portare il codice da un programma ad un altro. Non hai nemmeno bisogno di copiare ed incollare, basta usare le istruzioni per importare le funzioni e le variabili necessarie.

Con la OOP non devi creare un oggetto 'cane': infatti basta che cerchi qualcuno che l'abbia già realizzato e importarlo nei tuoi programmi.

OLTRE LA PROCEDURA

Quando incomincerai a programmare, lo farai scrivendo codice procedurale.

Nella buona vecchia programmazione procedurale di solito crei tutte le variabili all'inizio del programma. Poi definisci alcune funzioni (i blocchi di codice riutilizzabili).

Abbiamo già dato uno sguardo alla programmazione procedurale nel numero 53 di ***The MagPi*** (magpi.cc/2lin6PQ).

OOP utilizza le basi della programmazione procedurale – variabili, funzioni, cicli, condizioni – ma li accorpa in blocchi di codice indipendenti

Molti programmatori creano script procedurali che importano oggetti (da moduli o librerie). In questo modo usano gli oggetti senza nemmeno rendersene conto.

I concetti della OOP si possono trovare quasi in tutti i moderni linguaggi di programmazione inclusi Python e Java.

Importare la conoscenza

All'inizio di molti programmi trovi una serie di istruzioni per l'importazione. Queste sono utilizzate per includere il codice scritto da altre persone.

Il paradigma OOP non è perfetto. A volte è considerato eccessivo. "Il problema con i linguaggi orientati agli oggetti è che si portano dietro un ambiente implicito", dice Joe Armstrong, il creatore di Erlang. "Volevi una banana e ti ritrovi con un gorilla che regge una banana più tutto il resto della giungla."

Inoltre c'è tutto un lessico ornamentale che ruota attorno alla OOP. Incontrerai un sacco di strane parole come 'incapsulamento' ed 'istanziazione'. Queste fanno sembrare il concetto più complesso di quanto non sia in realtà, e possono essere piuttosto scoraggianti per i principianti.

Quindi la programmazione ad oggetti è piuttosto prolissa ed autocontemplativa. Molti maker, hacker e programmatori soffrono per comprendere la OOP anche se è possibile realizzare molte cose pur senza comprenderla.

D'altro canto i giovani programmatori sono sempre più spesso iniziati alla programmazione attraverso Scratch.

Scratch è incluso in Raspbian (ed in Debian con PIXEL) ed è stato creato per insegnare agli studenti la OOP, in modo trasparente.

In Scratch gli oggetti si chiamano 'Sprites' ed assomigliano ai personaggi dei videogiochi. L'idea è che i bambini che crescono con Scratch si sentiranno più a loro agio con gli oggetti quando incominceranno ad utilizzare linguaggi come Python.

CREA BUNCO IN SCRATCH

Crea un gioco in Scratch dove i giocatori possono giocare a dadi tra loro

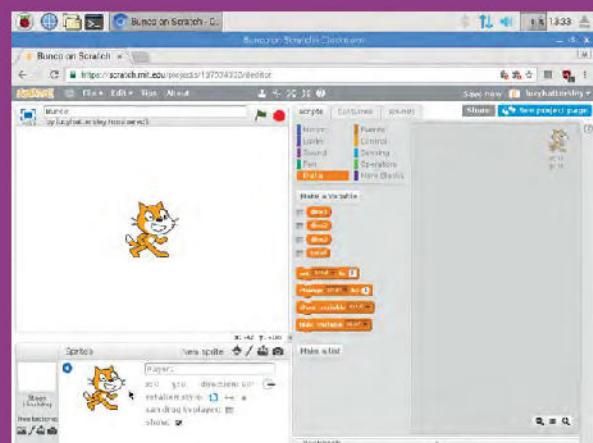
Cosa Serve

- Raspbian con PIXEL
- Un account Scratch 2.0

Prima creeremo un gioco in Scratch, poi in seguito lori faremo con Python in modo che tu veda come funziona con entrambi i linguaggi.

Il nostro gioco di dadi si basa su Bunco (**magpi.cc/2hoZNcj**), un gioco di società ,popolare in Nord America.

Abbiamo semplificato le regole. Ogni giocatore lancia tre dadi e conta i punti. Il giocatore col punteggio più alto vince.

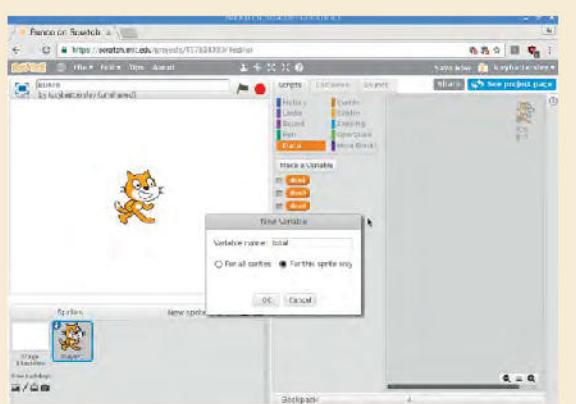
Abbiamo bisogno di due sprite (giocatori), ognuno col proprio set di dadi. Ogni giocatore lancia i dadi, controlla il punteggio e poi confronta i dadi con quelli dell'avversario.

Se i punteggi sono uguali entrambi dicono "pareggio". Se uno dei due si accorge che il suo punteggio totale è più alto di quello dell'altro giocatore, allora griderà "Ho vinto!".

Questo gioco servirà ad introdurti al concetto di variabili locali. Ciascun giocatore avrà tre variabili: il rispettivo set di dadi. I giocatori possono anche vedere le variabili (o dadi) che sono locali per l'altro.

L'opposto di una variabile locale è una variabile globale. Se usassimo una di queste sarebbe come se ciascun giocatore lanciasse lo stesso set di dadi e condividesse il risultato con l'altro. Sarebbe sempre pareggio.

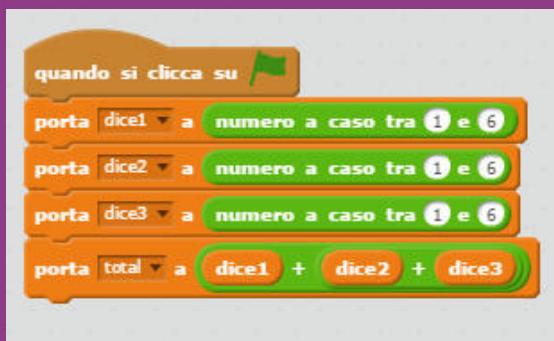
Scratch funziona in modo leggermente diverso da Python. In Scratch crei un personaggio con lo sprite e poi lo cloni (duplichi) per crearne un secondo. In Python tu crei un progetto per i tuoi personaggi (noto anche come 'classe') o poi costruisci due oggetti giocatore. Arriveremo a Python tra poco.

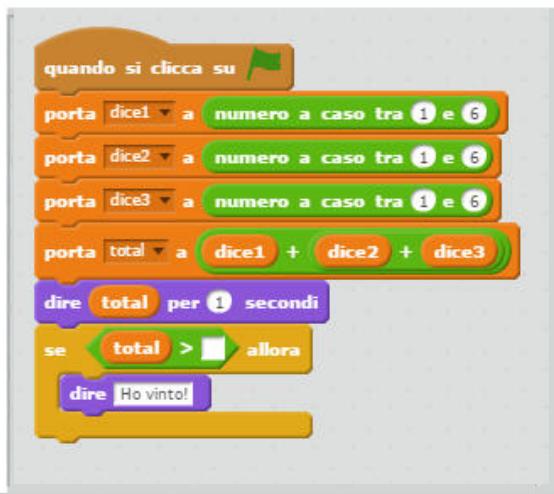
Prima, creiamo il gioco di dadi con Scratch...

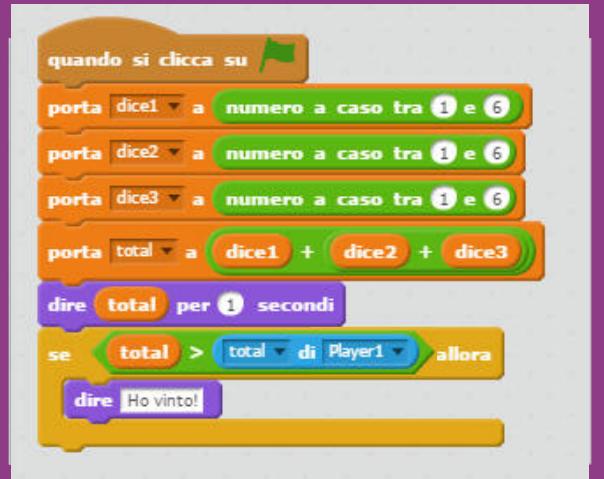
>PASSO-01

Scratch

Apri il browser e visita il sito ***scratch.mit.edu*** per aprire Scratch 2.0. Ci serve la funzione clone della versione 2.0, quindi non utilizzare la versione 1.4 dell'applicazione. Autenticati (oppure crea un account se sei nuovo in Scratch). Crea un nuovo progetto e vedrai lo sprite del gatto Scratch sullo schermo. Premi il simbolo 'i' sull'icona in basso a sinistra. Cambia il nome in Player1.

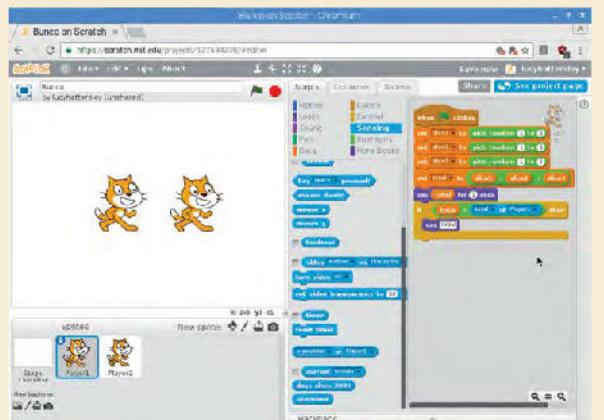
>PASSO-02

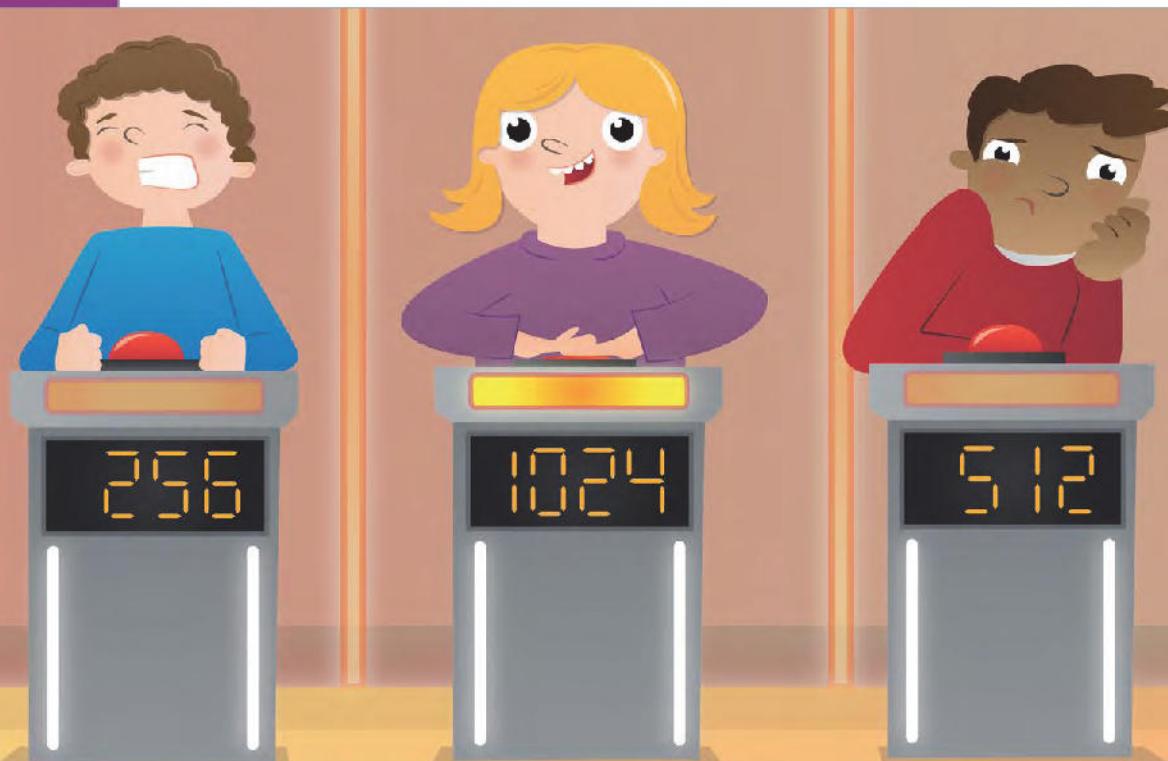
Tre dadi

Premi su Data e poi su Make a Variable. Digita **dice1** nel campo Name della variabile e seleziona l'opzione 'For this sprite only'. Premi OK e **dice1** apparirà nella tavolozza dei blocchi. Ripeti il procedimento per creare **dice2** e **dice3**. Infine crea un'altra variabile chiamata **total**. Ricordati di scegliere l'opzione 'For this sprite only' per tutti e tre i dadi e per il totale.

>PASSO-03
Lanciare i dadi

Clicca su Situazioni e trascina il blocco **quando si clicca su bandiera verde** nell'area dello script. Sotto a questo, devi aggiungere tre blocchi **porta diceN a numero a caso tra 1 e 6**. Sotto questi blocchi, aggiungi **sporta totale a dice1 + dice2 + dice3** (devi trascinare un blocco di addizione dentro l'altro per sommarli tutti e tre.

>PASSO-04
Parlare

Adesso trascina un blocco **dire per** e attaccalo alla fine del codice. Modificalo in **dire total per 1 secondi**. Sotto di esso, trascina un blocco **se**, dentro il quale, aggiungi un blocco **[] > []** (maggiore di). Trascina la variabile **total** sul lato sinistro del blocco maggiore di. Trascina un blocco **dire** all'interno del blocco **se** e cambia 'Hello!' in 'Ho vinto!'.

```
quando si clicca su
porta dice1 a numero a caso tra 1 e 6
porta dice2 a numero a caso tra 1 e 6
porta dice3 a numero a caso tra 1 e 6
porta total a dice1 + dice2 + dice3
dire total per 1 secondi
se total > [] allora
    dire Ho vinto!
```

>PASSO-05
Confrontare i punteggi

Finora, abbiamo un solo sprite. Ma stiamo per aggiungerne un altro e confrontare il totale di uno sprite con l'altro. Scegli i blocchi Sensori, e cercane uno che si chiama **posizione x di Player1⊠** Modifica '**posizione x**' con total. Trascina poi il blocco dal lato corretto del blocco maggiore di.

```
quando si clicca su
porta dice1 a numero a caso tra 1 e 6
porta dice2 a numero a caso tra 1 e 6
porta dice3 a numero a caso tra 1 e 6
porta total a dice1 + dice2 + dice3
dire total per 1 secondi
se total > total di Player1 allora
    dire Ho vinto!
```

>PASSO-06
Secondo giocatore

Il Tuo primo giocatore è pronto. Ora cloneremo lo sprite per creare il secondo giocatore. Tasto destro sullo sprite e scegli Duplica. Il nuovo sprite a forma di gatto, sarà automaticamente chiamato 'Player2'. Clicca su Player1 nella finestra degli sprite. Modifica il blocco **total di Player1** in **total di Player2** (come mostrato sotto). Ora il Player1 confronta il proprio totale con il Player2 (e il Player2 confronta il proprio con il Player1). Clicca sulla bandierina verde per lanciare il programma e vedere chi vince.

AMBITO

Il concetto di ambito è importante nella programmazione orientata agli oggetti. In Scratch, è così semplice che non te ne accorgi nemmeno. Ma i tuoi due giocatori hanno entrambi le loro variabili dice1, dice2 e dice3 più una total. Queste variabili sono in ambito locale. Quando il primo giocatore annuncia il totale, è il suo totale. Se entrambi gli sprite avessero accesso alla stessa total, sarebbe di ambito globale. Le Variabili negli oggetti sono in ambito locale.

CREA BUNCO IN PYTHON

Riscrivere il gioco Bunco in Python

La nostra semplice versione di Bunco gira bene in Scratch. Ora la ricreeremo in Python. La traduzione tra i due linguaggi ci aiuterà a capire bene come funzionano gli oggetti.

Prima di tutto pensiamo a come realizzare il gioco in maniera procedurale.

C'è un modulo chiamato **random** che possiamo utilizzare per generare numeri casuali; quindi dobbiamo importarlo. Poi possiamo creare una lista per ciascun giocatore ed usare la funzione **randint** per aggiungervi tre numeri casuali compresi tra uno e sei.

Possiamo utilizzare un costrutto **if else** con la funzione **sum()** per sommare i numeri di ciascun giocatore e quello col punteggio più alto vincerà la partita.

Digita il codice dal file **bunco_procedural.py** per provare il programma.

Ci sono due problemi in questa procedura. Bunco è molto più complesso nel mondo reale. Si gioca in sei round ed i giocatori raggiungono i 21 punti solo nel

CLASSI ED ISTANZE

Una delle grandi differenze tra creare sprite in Scratch ed oggetti in Python è che gli oggetti vengono creati utilizzando una classe, che agisce da modello/progetto per l'oggetto.

In Scratch crei gli sprite e poi li duplichi, e il secondo ha le stesse funzioni del primo ed il suo proprio set di variabili.

In Python (ed altri linguaggi di programmazione) le cose funzionano in modo leggermente diverso. Non crei gli oggetti direttamente, ma un modello per gli oggetti, chiamato 'classe', che in questo caso, non ha niente a che vedere con la scuola. Indica invece una categoria di oggetti simili. La classe è piuttosto simile ad un pianeta di 'Classe M' in Star Trek: anche se diversi, questi pianeti sono tutti simili alla terra.

Una volta creata la tua classe, la userai per creare gli oggetti, che vengono chiamati 'istanze' o 'istanze oggetto' e condividono proprietà simili; hanno le stesse variabili e le stesse funzioni (chiamate 'metodi'). In Scratch crei uno sprite e poi lo duplichi (per ottenere due sprite). In Python crei una definizione di classe, che poi utilizzi per creare due istanze dell'oggetto.

caso in cui lancino i dadi ed ottengano tre numeri uguali e corrispondenti al numero del round in corso (tre '1' nel primo round, tre '2' nel secondo e così via). Questo risultato è detto lanciare un 'Bunco'.

Noi non lo faremo così complesso, ma aggiungeremo un nuovo tipo di giocatori. Gli imbroglioni! Una canaglia ha lanciato i dadi; l'altro furfante ne cambia uno per ottenere un sei.

Poi giocheremo migliaia di partite e vedremo chi sarà il vincitore.

Nella programmazione procedurale ricreare questa complessità è estremamente difficile e richiede un cambio radicale nel nostro approccio a Bunco. La OOP è la risposta.

I vantaggi della OOP

Invece di creare all'inizio una lista di variabili per ogni giocatore, definiamo una classe chiamata **Player**.

Il codice contenuto nel file **bunco_oop.py** rappresenta un giocatore che lancia i dadi. In seguito lo useremo per creare due giocatori (vedi 'Classi ed Istanze').

Iniziamo importando il modulo **randint** come nel nostro codice procedurale.

Adesso proseguiamo definendo i nostri oggetti giocatore e per farlo creiamo una definizione di classe che appare così:

```
class Player:
```

All'interno della definizione della classe va identato il codice che rappresenta l'oggetto giocatore.

Nota bene che il nome della classe ha la lettera maiuscola e, rispetto alla definizione di una funzione, non usa le parentesi.

La prima cosa che dobbiamo aggiungere è una lista che contenga i dadi. Normalmente è sufficiente scrivere **dice = []**, ma se la definissimo così...

```
class Player:
    dice = []
```

...avremmo un problema. Questo codice equivale all'opzione 'For all sprites' presente in Scratch. Tutti i giocatori creati con questo codice condivideranno lo stesso gruppo di variabili ed otterranno lo stesso risultato, mentre noi vogliamo usare l'equivalente di 'For this sprite only'.

Per assicurarci che tutti i nostri giocatori abbiano il loro proprio gruppo di variabili, dobbiamo racchiudere la lista **dice = []** all'interno di una funzione particolare chiamata **__init__()**.

Ecco come appare il codice:

```
class Player:
    def__init__(self):
        self.dice = []
```

La funzione **__init__()** viene eseguita ogni volta che utilizzi una classe per creare un oggetto.

Questa funzione è nota come 'costruttore' o 'istanziatore'.

Quando useremo questa classe **Player** per creare gli oggetti giocatore, il codice **__init__()** verrà eseguito ogni volta.

Linguaggio
>PYTHON

SCARICA:
magpi.cc/2jxE6WJ

bunco_procedural.py

```
import random

player1_dice = []
player2_dice = []

for i in range(3):
  player1_dice.append(random.randint(1,6))
  player2_dice.append(random.randint(1,6))

print("Lancio Giocatore 1" + str(player1_dice))
print("Lancio Giocatore 2" + str(player2_dice))

if sum(player1_dice) == sum(player2_dice):
  print("Pari")
elif sum(player1_dice) > sum(player2_dice):
  print("Vince il Giocatore 1")
else:
  print("Vince il Giocatore 2")
```

bunco_oop.py

```
from random import randint

class Player:
  def __init__(self):
    self.dice = []

  def roll(self):
    self.dice = [] # clears current dice
    for i in range(3):
      self.dice.append(randint(1,6))

  def get_dice(self):
    return self.dice

player1 = Player()
player2 = Player()

player1.roll()
player2.roll()

print("Lancio Giocatore 1" + str(player1.get_dice()))
print("Lancio Giocatore 2" + str(player2.get_dice()))

if sum(player1.get_dice()) == sum(player2.get_dice()):
  print("Pari!")
elif sum(player1.get_dice()) > sum(player2.get_dice()):
  print("Vince il Giocatore 1!")
else:
  print("Vince il Giocatore 2!")
```

La funzione crea un set di dadi separato per ciascuno dei nostri giocatori.

Anche la parte 'self' ha bisogno di essere spiegata. Le variabili, come la nostra **dice = []** generalmente sono disponibili (o vengono restituite) quando una funzione termina.

Quindi se mettiamo **dice = []**, la lista verrà creata da **___init___()**, scomparendo subito dopo.

Python risolve questo problema con la parola chiave 'self' che devi inserire dentro le parentesi della funzione **___init___()**:

```
def __init__(self)
```

Poi useremo **self**, seguito da un punto, per memorizzare la variabile in questa versione dell'oggetto

```
self.dice = []
```

Poi userai **self.** all'interno delle funzioni, scrivendolo tra le parentesi, quando vorrai accedere alle variabili o cambiarle. In questo modo:

```
def roll(self):
```

Il concetto può risultare incomprensibile (passare una versione di se stesso dentro di se). Quindi focalizziamoci sul lato pratico invece che sulla teoria esoterica di come può funzionare:

- All'inizio di una classe metti la funzione speciale `__init__(self)`.
- Metti le variabili che vuoi usare dentro `init`.
- Crea le variabili usando `self.`, come `self.name` o `self.age` o `self.dice = []`.
- Inserisci `self` nelle parentesi delle funzioni che hanno bisogno di accedere alle variabili.
- Usa `self.` ed il nome della variabile per usarla dentro la funzione .

Ci siamo? Non preoccuparti se ti sembra strano, è la parte più difficile è sarà più facile con la pratica.

Adesso che abbiamo la nostra lista dei dadi, come facciamo con le altre funzioni?

Metodi tra la follia

Adesso che la nostra classe ha una lista per i dadi, abbiamo bisogno di lanciarli e per farlo definiremo una funzione più comune.

```
def roll(self):
  dice = [] # azzera il valore di dice
  for i in range(3):
     self.dice.append(randint(1,6))
```

Le funzioni degli oggetti si chiamano 'metodi', ma sono create nello stesso modo.

Ci sono parecchi tipi di metodo, puoi creare quelli che vuoi, ma i più comuni sono quelli chiamati 'setter' e 'getter'.

Il nostro metodo **roll** è un 'setter'. Imposta la lista **dice** usando tre numeri casuali.

Cosa pensi che faccia un 'getter'? Esattamente, legge il contenuto delle variabili di un oggetto e le restituisce.
Ecco qui il nostro getter:

```
def get_dice(self):
  return self.dice
```

I getter ed i setter all'inizio sembrano piuttosto strani. Dopo tutto puoi sempre recuperare l'oggetto per accedere alle variabili.

Almeno in Python questo è possibile, ma questa è considerata una cosa sbagliata. Uno dei cardini della OOP è che un oggetto contiene le sue variabili e le mantiene al sicuro dagli altri oggetti. Quindi non puoi semplicemente accedere ad un oggetto per leggerne le variabili.

Invece, crei dei metodi (funzioni) che settano le variabili o le leggono. Poi utilizzi questi metodi per settarle e leggerle.

Adesso che abbiamo creato la nostra definizione di classe, possiamo usarla per creare gli oggetti.

Creare lontano

Gli oggetti si creano esattamente come le variabili. Uso l'operatore di assegnazione (=). Adesso creiamo due giocatori che lanciano i dadi.

```
player1 = Player()
player2 = Player()
```

Nota bene che **player1** e **player2** non vengono chiamate 'variabili', ma 'istanze dell'oggetto'.

Accediamo ai metodi di un istanza di oggetto usando la notazione puntata, la sintassi in cui utilizzi il nome dell'istanza dell'oggetto seguita da un punto, seguito a sua volta dal nome del metodo che vorresti utilizzare.

Abbiamo creato un metodo, **get_dice()**, che restituisce il valore dei dadi memorizzato. Per accedere a questo metodo con la notazione puntata scriveremo **player1.get_dice()**.

Come prima cosa utilizziamo il metodo **roll** in modo che ciascun giocatore lanci i suoi dadi:

```
player1.roll()
player2.roll()
```

Il resto del nostro programma ***bunco_ oop.py*** è davvero molto simile al file ***bunco_procedural.py***. La differenza è che qui utilizziamo il metodo `.get_dice()` al posto di `sum()`.

Imbroglioni ereditari

Abbiamo già detto che uno dei vantaggi della OOP è la possibilità di creare centinaia di migliaia di giocatori con il loro set di variabili.

C'è una cosa che merita attenzione nel nostro programma, tutti i giocatori utilizzano gli stessi dadi ed hanno le stesse possibilità di vittoria. Ma cosa succederebbe se introducessimo dei bari?

Creiamo un imbroglione che gira uno dei dadi per ottenere un sei. Il tizio.

Il nostro imbroglione non è solo, ma ce n'è un altro che usa tre dadi truccati, che vincono sempre di un punto (a meno che non abbiano già dato sei).

Entrambi gli imbroglioni batterebbero un giocatore normale giocando poche centinaia di partite, ma quale degli imbroglioni batterebbe l'altro?

Non è così difficile trovare la risposta alla domanda, dobbiamo semplicemente simulare alcune centinaia di migliaia di partite.

Creeremo i nostri imbroglioni utilizzando una tecnica che si chiama ereditarietà. Si tratta della tecnica utilizzata per creare una classe che prenda le

> Ma quale trucco avrebbe vinto, contro l'altro?

caratteristiche (variabili di istanza e metodi) di un'altra, e ne aggiunge altre sue specifiche.

Pensa ad un bambino che eredita le caratteristiche dai propri genitori. Potrebbe avere il grande naso del padre ma crescere con le ginocchia bitorzolute, dovute tutte solo a lui.

I nostri imbroglioni ereditano le stesse funzioni **dice** e **roll** del 'parent' (classe padre), ma possiedono anche delle funzioni specifiche per imbrogliare.

Nel nostro programma **bunco_module.py** è definita la classe **Player()** ed i due figli:

```
class Player:
class Cheat_Swapper(Player):
class Cheat_Loaded_Dice(Player):
```

Gli oggetti che ereditano dal padre sono definiti usando lo stesso nome di classe.

Però in questo caso il nome della classe padre viene inserito tra le parentesi del figlio.

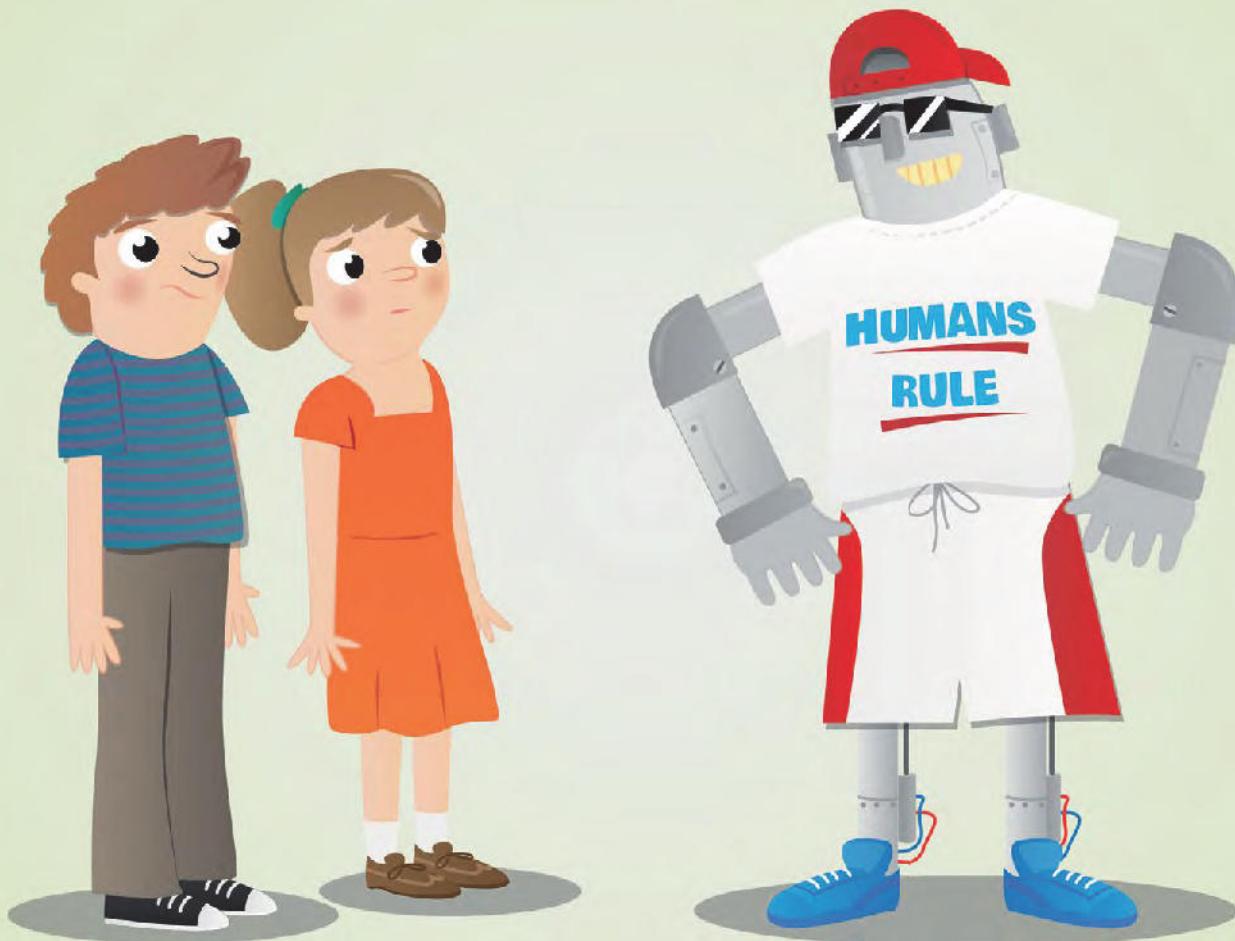
I nostri due imbroglioni ereditano tutte le variabili ed i metodi dal padre, quindi hanno già un lista **dice** ed i metodi **roll()** e **get_dice()**.

Adesso aggiungiamo a ciascuno di essi un nuovo metodo, che chiameremo **cheat**, implementato in maniera diversa per ciascun tipo di trucco.

bunco_module.py

```
from random import randint

class Player:
  def __init__(self):
    self.dice = []

  def roll(self):
    self.dice = [] # azzera i punteggi correnti
    for i in range(3):
      self.dice.append(randint(1,6))

  def get_dice(self):
    return self.dice

class Cheat_Swapper(Player):
  def cheat(self):
    self.dice[-1] = 6

class Cheat_Loaded_Dice(Player):
  def cheat(self):
    i = 0
    while i < len(self.dice):
      if self.dice[i] < 6:
        self.dice[i] += 1
      i += 1
```

bunco_single_test.py

```
from bunco_module import Player
from bunco_module import Cheat_Swapper
from bunco_module import Cheat_Loaded_Dice

cheater1 = Cheat_Swapper()
cheater2 = Cheat_Loaded_Dice()

cheater1.roll()
cheater2.roll()

cheater1.cheat()
cheater2.cheat()

print("Lancio imbroglione 1" + str(cheater1.get_dice()))
print("Lancio imbroglione 2" + str(cheater2.get_dice()))

if sum(cheater1.get_dice()) == sum(cheater2.get_dice()):
  print("Pari!")

elif sum(cheater1.get_dice()) > sum(cheater2.get_dice()):
  print("Vince imbroglione 1!")

else:
  print("Vince imbroglione 2!")
```

La definizione della classe **Cheat_Swapper** **contiene un metodo per barare piuttosto semplice:**

```
class Cheat_Swapper(Player):
  def cheat(self):
    self dice[-1] = 6
```

Il metodo per imbrogliare della classe **Cheat_Swapper** cerca l'ultimo elemento nella lista **dice** e lo imposta a 6.

La classe **Cheat_Loader_Dice** ha un metodo un po più complesso:

```
class Cheat_Loaded_Dice(Player):
  def cheat(self):
    i = 0
    while i < len(self.dice):
      if self.dice[i] < 6:
        self.dice[i] +=1
      i +=1
```

Questo metodo scorre i dadi presenti nella lista controllando se il risultato è minore di sei, nel qual caso incrementa il valore di uno.

Assicurati di aggiungere il codice nel file **bunco_module.py** e di salvalo con lo stesso nome.

Osserva questo codice, alla fine non ha traccia di programmazione procedurale. Il motivo è che lo importeremo (in questo modo vedrai cosa accade quando usi **import** nei tuoi programmi in Python).

Adesso creeremo il codice che utilizza questi oggetti in un file separato. Inserisci il codice dal listato **bunco_single_test.py** ed assicurati di salvarlo nella stessa cartella in cui si trova **bunco_module.py**.

La prima riga importa la definizione della classe **Player** dal nostro file **bunco_module.py**.

```
from bunco_module import Player
```

1000 punti se ti sei accorto che bunco_module è richiamato senza l'estensione '.ppy'. Questo è come puoi importare nel tuo programma del codice contenuto in altri file.

La riga **import Player** incolla il codice della definizione **Class Player** dal file **bunco_module.py**, ed equivale ad includere il codice nel tuo programma.

Se confronti questa riga con l'istruzione **from random import randint**, che si trova nel file **bunco_module.py** scoprirai che il principio alla base è lo stesso.

Importiamo anche le altre due definizioni di classe che abbiamo creato:

```
from bunco_module import Cheat_Swapper
from bunco_module import Cheat_Loaded_Dice
```

Il codice contenuto in **bunco_single_test.py** crea lo stesso gioco presente nel file **bunco_oop.py** di prima.
Adesso creeremo due istanze oggetto utilizzando le classi **Cheat_Swapper** e **Cheat_loaded_Dice** importate da **bunco_module**:

```
cheater1 = Cheat_Swapper()
cheater1 = Cheat_Loaded_Dice()
```

Poi useremo il metodo **roll()**, da notare che né **Cheat_Swapper()** né **Cheat_Loaded_Dice()** hanno definito il metodo **roll**. Poiché ereditano questa funzione dalla classe padre, **Player()**:

```
cheater1.roll()
cheater2.roll()
```

Poi chiameremo il metodo **cheat()** per entrambi gli oggetti:

```
cheater1.cheat()
cheater2.cheat()
```

Nonostante ogni oggetto abbia un metodo chiamato **cheat**, gli oggetti hanno una implementazione differente. Infatti **cheater1** cambia l'ultimo dado in un 6, mentre **cheater2** incrementa il valore di ciascun dado di 1.
Esegui il programma premendo **F5** e verifica chi vince. Se lo esegui un altra volta otterrai un risultato diverso. Continua ad eseguirlo e ti accorgerai che si tratta di un testa a testa.
Se guardi nella cartella che contiene i sorgenti ti accorgerai che è comparso un nuovo file chiamato **bunco_module.pyc**. Si tratta di un 'file compilato' che viene creato la prima volta che esegui un programma che importa del codice. Generalmente non guardi i file compilati perché importi del codice che è nascosto nei meandri di Python nel tuo computer. No ti preoccupare di questo, tanto si tratta di file che non puoi aprire in un editor di testo. Se vuoi lo puoi anche cancellare perché verrà ricreato quando utilizzerai il file **bunco_module.py** nella versione finale del programma. Per ora lo puoi semplicemente ignorare.
Per scoprire quale dei due imbroglioni ha la meglio sull'altro dobbiamo eseguire una simulazione. Dobbiamo giocare centinaia di migliaia di partite tenendo traccia di chi vince.
Il programma definitivo, **bunco_simulation.py**, fa esattamente questo. Questo programma mette insieme tutto quello che abbiamo imparato sulla OOP. Il codice contenuto nel file **bunco_simulation.py** crea due imbroglioni e gioca 100.000 partite. Importa le definizioni di classe dal nostro programma **bunco_module.py** (quindi assicurati di averlo salvato nella stessa cartella).

bunco_simulation.py

```
from bunco_module import *

swapper = Cheat_Swapper()
loaded_dice = Cheat_Loaded_Dice()


swapper_score = 0
loaded_dice_score = 0

number_of_games = 100000
game_number = 0

print("Simulazione in corso")
print("==================")
while game_number < number_of_games:
  swapper.roll()
  loaded_dice.roll()

  swapper.cheat()
  loaded_dice.cheat()

  #Rimuovi il # prima delle istruzioni print per attivare la
  #Simulazione. Impiegherà circa un'ora con le istruzioni print
  #oppure dieci secondi con le istruzioni print commentate


  #print("Cheater 1 rolled" + str(swapper.get_dice()))
  #print("Cheater 2 rolled" + str(loaded_dice.get_dice()))

  if sum(swapper.get_dice()) == sum(loaded_dice.get_dice()):
    #print("Pari!")
    pass

  elif sum(swapper.get_dice()) > sum(loaded_dice.get_dice()):
    #print("Vince Dice Swapper!")
    swapper_score+= 1

  else:
    #print("Vince Loaded Dice!")
    loaded_dice_score += 1

  game_number += 1

print("Simulazione completata")
print("-------------------")
print("Punteggi finali")
print("------------")
print("Swapper ha vinto: " + str(swapper_score))
print("Loaded Dice ha vinto: " + str(loaded_dice_score))

if swapper_score == loaded_dice_score:
  print("Il gioco è finito in parità")
elif swapper_score > loaded_dice_score:
  print("Swapper ha vinto più partite")
else:
  print("Loaded Dice ha vinto più partite")
```